National Defence / Défense nationale

DESCRIPTION OF AN ARINC BUS INTERFACE FOR AN LSI-11 MINICOMPUTER

by

Marc Dion
Electromagnetic Section
Electronic Division

DEFENCE RESEARCH ESTABLISHMENT OTTAWA
TECHNICAL NOTE 88-30

PCN
041LJ

December 1988
Ottawa

ABSTRACT

An interface between the ARINC bus and an LSI-11 minicomputer was designed and built at DREO. The interface accepts serial data from a single twisted pair of cables which meet the ARINC 429 interface protocol specifications. The data is transferred in the minicomputer memory through a DEC standard DRV11 parallel interface. An interface driver is provided to control the operation of the interface.

RÉSUMÉ

Un interface entre le bus ARINC et un mini-ordinateur LSI-11 a été conçu et assemblé au CRDO. Il accepte des données en série respectant les spécifications d'interface ARINC 429. Un interface standard (parallèle) DRV11 de DEC est utilisé pour transférer les données au mini-ordinateur. Un logiciel de controle de l'interface est fourni.

EXECUTIVE SUMMARY

An interface was designed and built at the Defence Research Establishment Ottawa (DREO) to receive serial data from the ARINC 429 bus and to transfer it to an LSI-11 minicomputer. The interface was part of an elaborate set-up for testing inertial navigation systems and used to record the data coming from the navigation system under test.

This Technical Note gives a description of the ARINC 429 bus, the interface board, the DRV11 interface and the related software. The interface board schematic diagrams and complete listings of the software are also included.

The interface accepts serial data from a single twisted pair of cables which meet the ARINC 429 interface protocol specifications. The ARINC 429 bus specification defines a communication protocol for the transfer of digital data between avionics systems elements.

The interface board was specially designed at DREO for this application. It converts the serial data into a parallel format and stores it into a buffer. Buffering is necessary to accommodate for the different speed of operation of the ARINC bus and the minicomputer. The data is then transferred to the minicomputer memory through a DEC standard DRV11 parallel interface. All operations of the interface are under program control.

Software is provided to control the operation of the interface. It is optimised for minimal overhead and is designed to interface with high-level languages such as FORTRAN. It includes several subroutines to facilitate the use of the interface. They perform the initialisation of the interface, the selection and acquisition of the data and the necessary buffer management.

TABLE OF CONTENTS

		PAGE
ABSTRACT/RÉSUMÉ		iii
EXECUTIVE SUMMARY		v
TABLE OF CONTENTS		vii
1.0	GENERAL INFORMATION	1
1.1	GENERAL DESCRIPTION	1
1.2	RELATED DOCUMENTATION	1
1.3	CONVENTIONS	1
2.0	THE ARINC 429 BUS	4
2.1	ELECTRICAL CHARACTERISTICS	4
2.2	TRANSMISSION FORMAT	4
2.3	DATA RATES	4
3.0	LSI-11 DRV11 INTERFACE	7
3.1	CONFIGURATION	7
3.2	CONTROL/STATUS REGISTER	7
3.3	INPUT DATA BUFFER	8
4.0	INTERFACE DESCRIPTION	9
4.1	LINE RECEIVERS	9
4.2	ARINC-TO-PARALLEL CONVERSION	9
4.3	FIFO BUFFERS	9
4.4	SEQUENCER AND CONTROL LOGIC	9
4.5	INTERRUPT LOGIC	10
4.6	OSCILLATOR	10
4.7	POWER SUPPLY	10

5.0 SOFTWARE DESCRIPTION 13

5.1 CONFIGURATION AND SET-UP 13

5.2 GENERAL DESCRIPTION 14

5.3 SUBROUTINE DESCRIPTION 14

5.3.1 Subroutine DRINIT 15

5.3.2 Subroutine DRDISC 15

5.3.3 Subroutine DRSEL 16

5.3.4 Subroutine DRSTRT 16

5.3.5 Subroutine DRSTOP 16

5.3.6 Subroutines DRRESM and DRSPND 17

5.3.7 Subroutine DRGET 17

APPENDIX A ..... INTERFACE SCHEMATIC DIAGRAMS 20

APPENDIX B ..... INTERFACE SOFTWARE 26

TABLES

TABLE 2-1 ARINC 429 VOLTAGE LEVELS 5

TABLE 5-1 STATUS CODES 19

FIGURES

FIGURE 1-1 BLOCK DIAGRAM 3

FIGURE 2-1 ARINC 429 WORD FORMAT 5

FIGURE 2-2 ARINC 429 TIMING DIAGRAM 6

FIGURE 3-1 CSR FORMAT 7

FIGURE 3-2 INBUF FORMAT 8

FIGURE 4-1 IC #575042 TIMING DIAGRAM 11

FIGURE 4-2 SEQUENCER TIMING DIAGRAM 12

FIGURE 5-1 DRGET WORD FORMAT 19

APPENDIX A INTERFACE SCHEMATIC DIAGRAMS 20

1.0 INTRODUCTION

An interface was designed and built at the Defence Research Establishment Ottawa (DREO) to receive serial data from the ARINC 429 bus and to transfer it to an LSI-11 minicomputer. The interface accepts serial data from a single twisted pair of cables which meet the ARINC 429 interface protocol specifications. The data is transferred to the minicomputer memory through a DEC standard DRV11 parallel interface. An interface driver is provided to control the operation of the interface. This driver is optimised to minimise interrupt processing time and operating system overhead and is designed to interface with high-level languages such as Fortran.

This Technical Note gives a description of the ARINC 429 bus, the interface board, the DRV11 interface and the interface driver. The interface board schematic diagrams and complete listings of the interface driver are included in the Appendix.

1.1 GENERAL DESCRIPTION

The Figure 1.1 shows a block diagram of the system. The information sent serially onto the ARINC data bus by some navigational system or other equipment via a single pair of cables feeds the interface board. This interface board was specially designed at DREO for this application. It converts the ARINC data into a parallel format and stores it into a first-in, first-out buffer. The data is then transferred into the LSI-11 minicomputer memory under the control of the minicomputer. The interface board connects directly to a standard DRV11 Q-bus interface. An interface driver was also developed to control the hardware and provide a software interface to the application task. The DRV11, the driver and the application task reside on the LSI-11 minicomputer.

1.2 RELATED DOCUMENTATION

- ARINC Specification 429-7, April 1983
- PDP-11 Microcomputer Interfaces Handbook, 1983
- RSX-11M Documentation Set
- PDP-11 Fortran-77 Reference Manuals

1.3 CONVENTIONS

Numbers used in this document are naturally in their decimal form, but will sometime be printed in their octal form to be consistent with the associated literature or related software.

Bits within a word are numbered from 0, starting with the

least significant bit. The only exception, is the definition of the ARINC data words and fields which are numbered from 1 to remain consistent with the ARINC specification.

Signal names which are overscored or followed by an asterisk (*) for signals which are level significant denotes that the signal is true or valid when the signal is low.

Signal names which are overscored or followed by an asterisk (*) for signals which are edge significant denotes that the actions initiated by that signal occur on a high to low transition.

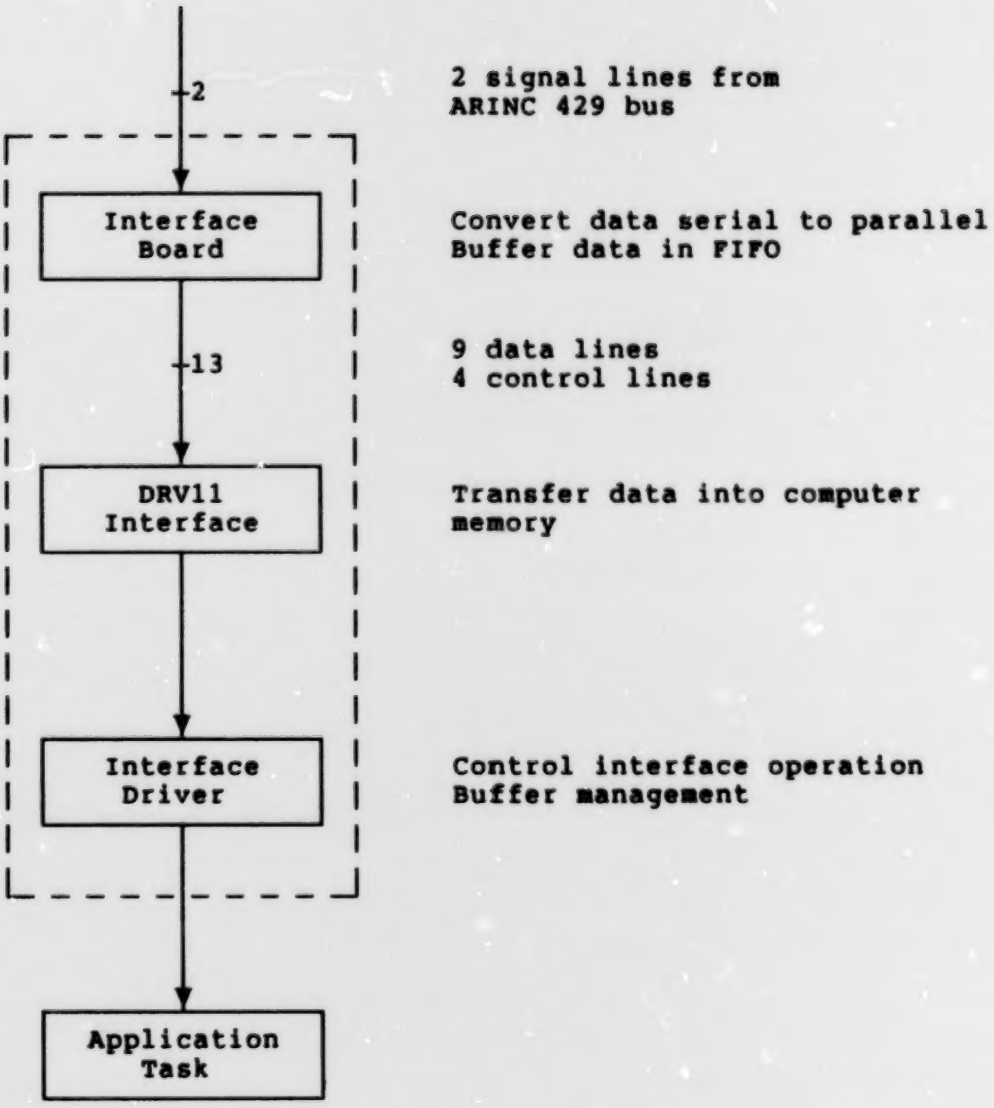


Figure 1-1 Block Diagram

2.0 THE ARINC 429 INTERFACE

The ARINC 429-7 bus specification defines a communication protocol for the transfer of digital data between avionics systems elements. It was first adopted by the Aeronautical Radio Inc (ARINC) in July 1977. The current ARINC 429-7 specification was adopted in November 1982. the standard specifies both the electrical limits of the signal and the encoding rules for the data.

2.1 ELECTRICAL CHARACTERISTICS

The data is transmitted by means of a single twisted and shielded pair of wires. The data is encoded with a tri-level state modulation consisting of "HI", "NULL" and "LO" states. Differential output signals are used. Figure 2-2 shows the differential voltage on the bus in relation with the binary data generated. Table 2-1 gives the nominal voltages at the output of the transmitter.

2.2 TRANSMISSION FORMAT

All data transfers consist of one or more 32-bit words. Each word contains a label, a sign/status matrix, a data field and a parity bit. The ARINC 429 completely specifies the content of each field. The label field (8 bits) identifies the type of data being transferred. The sign/status matrix (2 bits) is used to implement higher level of protocols. The data field is 21 bits wide and its format depends on the label field. The last bit is used to render word parity odd. The interpretation of the sign/status matrix and the data field is left to the application task. Figure 2-1 shows the ARINC 429 word format.

Words are transmitted serially, with the least significant bit transmitted first. The least significant bit of each field is transmitted first with the exception of the label field for which the most significant bit is transmitted first (reverse label). A gap between words of at least 4 bit periods is required for synchronisation.

2.3 DATA RATES

The data transmitted onto the ARINC bus, identified by the label field, may include: position (latitude and longitude), heading, speed, acceleration, rates. This information is updated at a specified interval, usually 1, 2, 4, 8, 16, 32 or 64 Hz. This translates to an average data rate of approximately 1000 words/second (4 kByte/sec) for a particular system. A mini-computer in a multi-tasking environment could handle those data rates without significantly degrading the overall system performances. However, the data on the ARINC bus often comes in

burst at speeds which exceed the capacity of the minicomputer unless some buffering is provided. The interface board was designed to provide such buffer to achieve good performance.

TABLE 2-1. ARINC 429 Voltage Levels (Volts)

	HI	NULL	LO
Line A to Line B	+10 ±1.0	0 ±0.5	-10 ±1.0
Line A to Ground	+5 ±0.5	0 ±0.25	-5 ±0.5
Line B to Ground	+5 ±0.5	0 ±0.25	-5 ±0.5

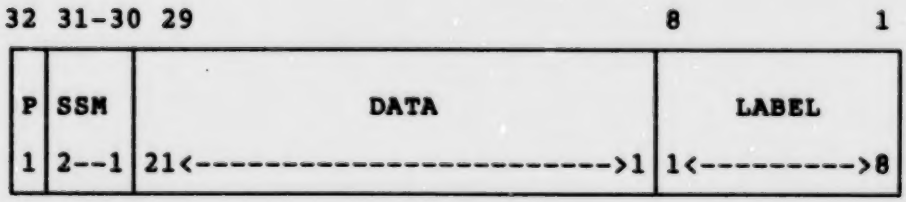


FIGURE 2-1. ARINC 429 Word Format

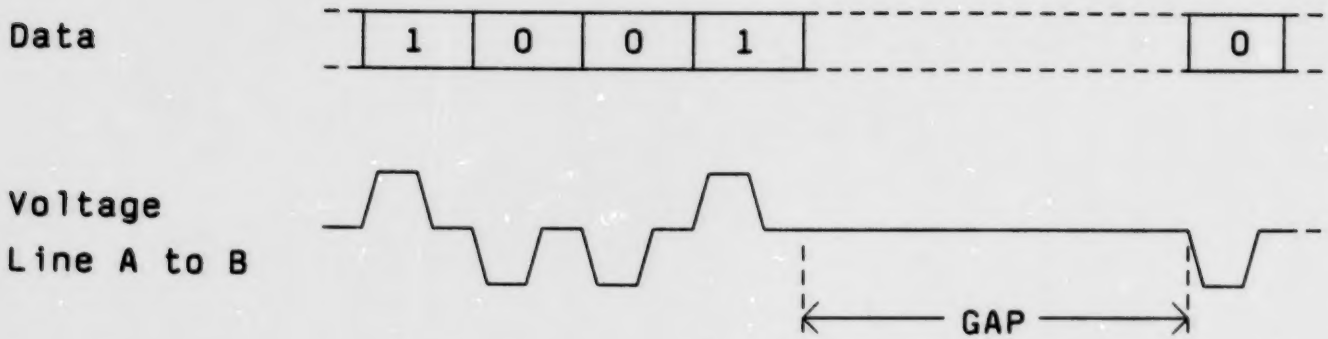


Figure 2-2. ARINC 429 Timing Diagram

3.0 LSI-11 DRV11 INTERFACE

The DRV11 is a general purpose interface unit used for connecting parallel TTL lines to the LSI-11 bus (Q-bus). The DRV11 is controlled through a control/status register (CSR) and 2 data registers (INBUF and OUTBUF). It provides 16 input data lines, 16 output data lines and 4 control lines, including provision for interrupt request. Only 10 input data lines and 4 control lines are used.

3.1 CONFIGURATION

The DRV11 module must be configured with a proper address and interrupt vector as described in its manual. The current setting is 170200, 150. The interface driver, described in chapter 5, must be built with the selected settings.

The ARINC interface board is connected to the input port (J2) of the DRV11 board with a shielded 40-conductor flat cable. The shielding is grounded at the DRV11 end only. The output port, accessed through J1, is not used and could be used by other applications, providing INBUF is not accessed and that bits INT_ENA_B and CSR0 in the CSR are not changed.

3.2 CONTROL/STATUS REGISTER

The control/status register controls the overall operation of the module. Only 3 bits are used to control the operation of the interface board: INT_ENA_B is used to enable the interrupts, REQ_B is used as an interrupt request and CSR0 is used to reset the interface board. Figure 3-1 shows the CSR format.

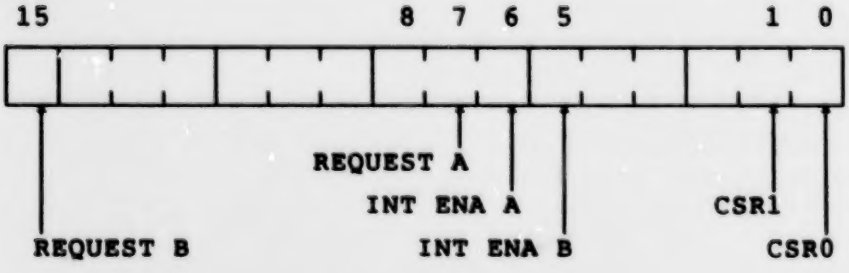


Figure 3-1. CSR Format

3.3 INPUT DATA BUFFER

When a 32-bit word is received from the ARINC bus, it is stored in a FIFO buffer as 4 consecutive bytes. Each time the input data buffer (INBUF) is read, one byte of data is retrieved along with a tag to identify the labels (LABEL) and one status bit (FIFO_EMPTY) to indicate that the FIFO buffer is now empty. Figure 3-2 shows the INBUF format.

When accessed, the DATA_TRANS signal on J2 is pulsed to allow synchronisation with the external logic.

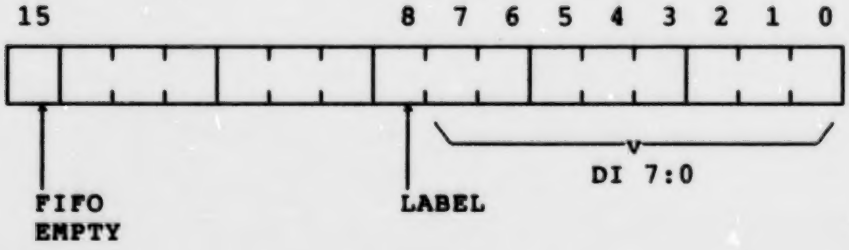


Figure 3-2. INBUF Format

4.0 INTERFACE DESCRIPTION

The interface board receives the incoming ARINC bus data, converts it to a parallel format, stores it into a first-in first-out (FIFO) buffer and delivers it to the DRV11 interface. A sequencer controls the operation of the interface. The complete schematic diagrams of the interface are given in Appendix A.

4.1 LINE RECEIVERS

The data from the ARINC bus is encoded with a tri-level state modulation by using a differential voltage. This signal is converted to TTL levels to form two binary signals (ARINC0 and ARINC1) by using analog-input comparators with TTL-output. Figure 4-1 shows the effect of the line receivers.

4.2 ARINC-TO-PARALLEL CONVERSION

A single integrated circuit (IC) manufactured by Litton (part #575042) converts a 32-bit serial data word into four 8 bit words, which can be accessed through four read operations. A timing diagram for this IC is given in Figure 4-1. A clock is required for the circuit operation and as a timer to detect gaps between words. A clock of 650 kHz is used, but it can be adjusted to different data rates.

4.3 FIFO BUFFERS

A pair of first-in, first-out memory (Mostek part #MK4501N-15) is used to provide 1024x9 bits of buffer memory. This IC features asynchronous read and write operations which considerably simplify the interface design. The data is written into it under the control of the sequencer and it is read under the control of the DRV11 interface. Simple logic is used to flag empty, half-full and full conditions.

4.4 SEQUENCER AND CONTROL LOGIC

A sequencer is used to detect the arrival of a 32-bit word, read 4 bytes from the Litton IC and store them into the FIFO. A flag (LABEL) is also generated and stored along the data to identify label fields. Figure 4-2 shows the timing diagram for one complete cycle.

The heart of the sequencer is a 74LS163 configured to count from 0 to 15 once. A counting cycle is initiated when a 32-bit word is received (FIFO_RDY* asserted). A synchronous pulse (START) is generated, which starts the counter. The outputs of the counter are used in conjunction with H* to generate 4 pulses

(ARINC_RD* and FIFO_WR*) to read the data from U11 and write it into the buffers (U9 and U10). A signal (LABEL) is also asserted during the first pulse and stored along with the data to identify the label field. The logic that generates FIFO_WR* (see U5A) was carefully designed to avoid any glitch on it. It is essential that the signal H* leads the clock H, but otherwise the logic does not rely on any delay to properly operate.

4.5 INTERRUPT LOGIC

The half-full flag is used to interrupt the computer through the REQ_B line. To be acknowledged, the INT_ENA_B bit in the CSR must be set. Data from the FIFO can then be read, until the empty flag (INBUF, bit 15) is detected.

4.6 OSCILLATOR

Three Schmitt-trigger inverters and a simple R-C network are used to generate a clock signal (H). The resistor is adjusted to set the frequency to 650 kHz. The exact frequency is not critical. A second clock is generated (H*) which is guarantied to be leading H due to the delay in the inverter U4E.

4.7 POWER SUPPLY

The interface board uses a single external 5V regulated power supply.

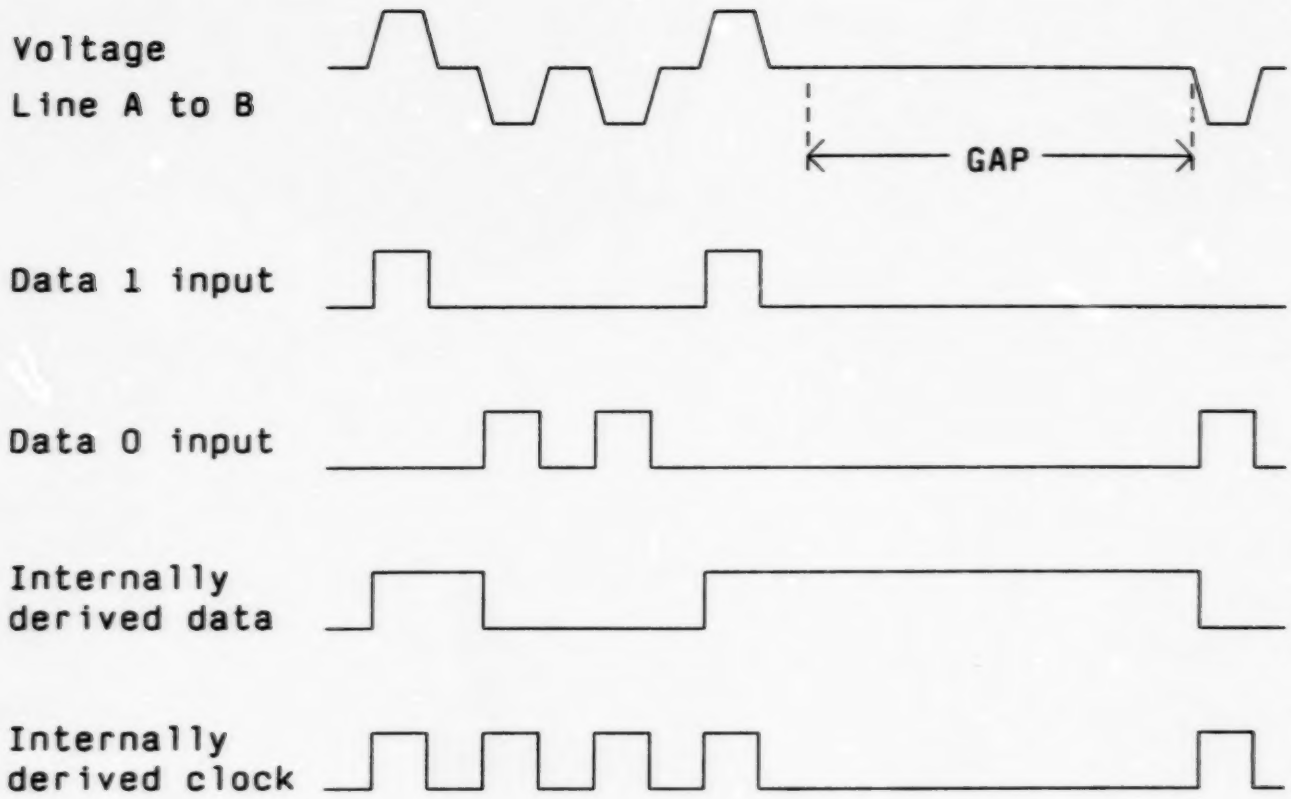


Figure 4-1. IC #575042 Timing Diagram

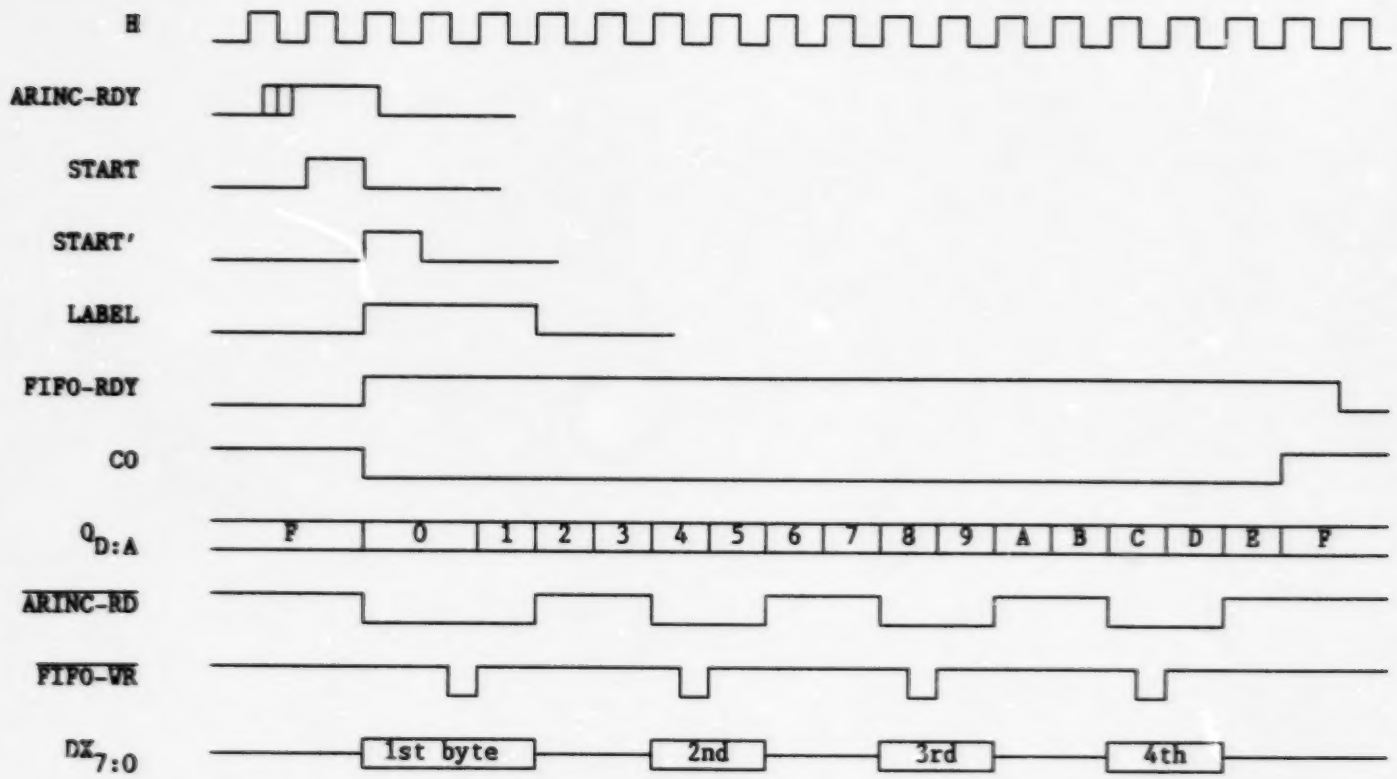


Figure 4-2. Sequencer Timing Diagram

5.0 SOFTWARE DESCRIPTION

A software module, referred to as the interface driver, was designed to control the overall operation of the ARINC interface through the DRV11 board. It contains several subroutines which can be called from high-level languages such as Fortran. The driver is included in the application task, which must be built privileged but not mapped onto the executive. A "connect to interrupt" directive is used to process the interrupts. Appendix B gives the complete listing of the interface driver.

5.1 CONFIGURATION AND SET-UP

In order for the application task to work successfully with the driver and the interface, the following requirements must be met.

- The address and vector of the DRV11 board should be selected and must not conflict with any other interface on the system.

- The following files must be accessible on the system:
  LB:[1,1]EXEMC.MLB
  LB:[11,10]RSXMC.MAC
  LB:[1,54]RSX11M.STB

- The device common IOPAGE must be installed on the system. It is a 4 kBytes partition which is mapped onto the I/O page of the Q-bus.

- The application task must be built (with TKB) with the following options (highlighted text is mandatory unless otherwise specified):

```
TKB>main/PR:0/-SG/-SQ/FP, main/-SP= main
TKB>sub1, sub2, subn
TKB>ARINCDR11, ARINCLUT
TKB>LB:[1,54]RSX11M.STB/SS
TKB>/
TKB>COMMON=IOPAGE:RW
TKB>EXTSCT=INTRP3:10000          (optional)
TKB>GBLDEF=DR$CSR:170200         Set to DRV11 address
TKB>GBLDEF=DR$VEC:150              and vector
TKB>//
```

The driver provides an initial buffer pool area of 1024 bytes. The EXTSCT option can be used to increase its size.

The GBLDEF options are used to specify the DRV11 address and vector.

5.2 GENERAL DESCRIPTION

The quantities transmitted on the ARINC bus are updated at periodic intervals. The update rate is usually a power of 2 (eg. 1, 8 or 64 Hz). However, the time of day is not transmitted with the data and therefore the driver relies on counting the labels to synchronise. One arbitrary label, which should be updated at the fastest rate (usually 64 Hz), is used as a time marker. That label is called the **SYNC** and the time interval (1/64 Hz) called a **TIC**.

The user can specify the labels of interest. Only those selected labels and their associated data will be stored in buffers and delivered to the user. A **FRAME** consists of all the selected labels (and data) received since the last tic and therefore the SYNC marks the beginning of a new frame. As frames are being received, they update the content of the current **BUFFER**. Buffers are created dynamically from a buffer **POOL** area. When a buffer is created, it is first initialised with the previous buffer. This ensures that the current buffer will contain the latest update of the data in the case some data were not updated in the last frame.

The driver can operate in two modes: **PERIODIC** and **APERIODIC**. In the aperiodic mode, the current buffer is continuously updated with the incoming data. When the user requests the data (through DRSEL), the current buffer is passed (by pointer) to the user and a new buffer is created. In the periodic mode, the current buffer is updated with the incoming data for a specified interval (**RATE**), given in tics. At the end of the interval, a new buffer is automatically created and becomes the current buffer. Previous buffers are kept until requested by the user (through DRGET). The buffers are delivered to the user in a FIFO fashion. An overrun condition may exist if the user does not read the buffers fast enough.

All buffers created from the pool area and returned to the user (by DRGET) reside in the program section INTRP3, which can be accessed from a Fortran program through a COMMON block. All the data is returned to the user in the form of a pointer in the COMMON block. Access to individual bytes is done with conventional indexing. Section 5.3.7 describes in detail the format and usage of the COMMON block.

5.3 SUBROUTINE DESCRIPTION

This section describes briefly each subroutine. Listings given in Appendix B are well documented if more details are necessary.

The PDP-11 Fortran-77 calling conventions are used, in which arguments are passed by reference (ie. by pointer). All arguments and results, unless otherwise specified, are in

"INTEGER*2" format. All subroutines return a status. Table 5-1 lists the status codes and their description.

5.3.1 Subroutine DRINIT

The subroutine DRINIT is called to initialised the interface driver and the interface board. A "connect to interrupt" directive is issued, but the interrupts are not enabled yet. Its only input argument is an event flag number which should be valid and reserved for the driver usage.

This subroutine has two more hidden arguments, which are the two symbols DR$CSR and DR$VEC which must be defined during task build. If the address specified by DR$CSR is non-existent, a trap #4 condition will occur and the task will be aborted. If the vector DR$VEC is invalid or in use, the CINT$ directive will fail.

Because of some limitation of the CINT$ directive, all the driver code and buffers must reside within a 8 kBytes area. This condition may not be met if the pool area is too big, as specified by the EXTSCT option. An error status IE.SPC will be returned in those conditions.

Fortran call: CALL DRINIT (EFL, IDS)

Arguments: EFL — Event flag number used for synchronisation.

Results: IDS — Status:
- IS.SUC - Success
- IE.ITS - Invalid call
- IE.IEF - Invalid event flag
- IE.SPC - Poll area too big
- IE.RSU, IE.PRI & others - as returned by the $CINT directive

5.3.2 Subroutine DRDISC

The subroutine DRDISC terminates all operations. It should be called before the user task terminate but would be automatically called otherwise.

Fortran call: CALL DRDISC (IDS)

Results: IDS — Status:
- IS.SUC - Success
- IE.ITS - Invalid call

5.3.3 Subroutine DRSEL

This subroutine is used to select the labels of interest. It must be called after DRINIT and before DRSTRT. The input arguments are a byte array and its size. The array contains a list of the labels. The ARINC data will be delivered to the user in the order specified in the list.

Fortran call:	CALL DRSEL (LABELS, N, IDS)	
Arguments:	LABELS	Array of bytes, size N List of labels of interest.
	N	Size of the array LABELS
Results:	IDS	Status: IS.SUC - Success IE.ITS - Invalid call

5.3.4 Subroutine DRSTRT

The subroutine DRSTRT is called to enable data acquisition. It first initialises the buffer area and then enables the DRV11 interface. It then waits for the SYNC label and synchronises onto it. After that, at least 8 frames are read to set up the first buffer. The control is returned to the caller with the driver in the aperiodic mode. A subsequent call to DRRESM will place the driver in periodic mode.

Two arguments are used to specify RATE and SYNC. SYNC is the label used as a time marker. RATE specifies the rate at which buffers will be automatically created in periodic mode. It also specifies the maximum rate at which buffers can be delivered to the user in aperiodic mode.

Fortran call:	CALL DRSTRT (RATE, SYNC, IDS)	
Arguments:	RATE	To specify the rate at witch buffers are created. Unit in 1/64 second.
	SYNC	Label used as a time marker (SYNC).
Results:	IDS	Status: IS.SUC - Success IE.ITS - Invalid call

5.3.5 Subroutine DRSTOP

The subroutine DRSTOP terminates the data acquisition but does not disconnect from the interface. It must be called if a different selection of labels (DRSEL) is wanted or if it is necessary to restart operations.

```
Fortran call:  CALL DRSTOP ( IDS )

Results:       IDS      Status: IS.SUC - Success
                                IE.ITS - Invalid call
```

5.3.6 Subroutines DRRESM and DRSPND

After the subroutine DRSTRT is called, the driver operates in aperiodic mode. Subroutine DRRESM is used to switch from aperiodic to periodic mode and subroutine DRSPND switches it back to aperiodic.

```
Fortran call:  CALL DRRESM ( IDS )

Results:       IDS      Status: IS.SUC - Success
                                IE.ITS - Invalid call

Fortran call:  CALL DRSPND ( IDS )

Results:       IDS      Status: IS.SUC - Success
                                IE.ITS - Invalid call
```

5.3.7 Subroutine DRGET

This subroutine is called to return to the user one buffer of ARINC data. The task will be suspended if the data is not already in the pool area. The subroutine returns a pointer to the buffer. A Fortran program can access the buffer through a COMMON statement as shown below. The buffer consists of ISIZ consecutive bytes in BUF starting at BUF(IDX) and ending at BUF(IDX+ISIZ). The user can read and modify the buffer returned by DRGET, but it is very important not to access other parts of BUF as it is used by the interrupt routine. The size of the buffer should be the number of selected labels times 4.

```
Define buffer: COMMON /INTRP3/ BUF
               BYTE            BUF(1)

Fortran call:  CALL DRGET ( IDX, ISIZ, IDS )

Results:       IDX      Index of buffer in INTRP3, in bytes
               ISIZ     Size of the buffer returned
               IDS      Status: IS.SUC - Success
                                IE.ITS - Invalid call
                                IE.DAO - Buffer overflow
                                IE.TMO - Time-out occurred
```

The order at which the ARINC data is received (see Figure 2-2) is rearrange before it is passed to the user. Figure 5-1 shows the format of the 32-bit words returned by DRGET. Bit 32 (P) is always set to 0. The LABEL field is reversed to put it back in the normal position and placed at the high end of a 32-bit word. For instance, if the array BUFW is defined as follow:

```
COMMON /INTRP3/ BUFW
BYTE            BUFW(4,*)
```

And DRGET returns a pointer in IDX, each individual byte in each 32-bit words is accessed as follow:

```
BUFW(i,n+IDX/4)
```

for n = 1 to N	Index BUFW to access any of the ARINC data words in the buffer returned by DRGET. They are in the same order as selected by DRSEL.
for i = 1	To access the ARINC data field, bits 1 to 8.
for i = 2	To access the ARINC data field, bits 9 to 16.
for i = 3	To access the ARINC data field, bits 17 to 21 and the SSM.
for i = 4	To access the ARINC label field.

The labels returned in the buffer match those specified with DRSEL, with one exception. If a particular label was never received since the data acquisition was started by DRSTRT, the label and data fields will be returned as zero.

TABLE 5-1. Status Codes

CODE	VALUE	DESCRIPTION	FROM
ISSUS	+1	Success	All
IESPC	-6	Illegal user buffer	DRINIT
IEITS	-8	Calls out of sequence	All
IEDAO	-13	Data overrun, data lost	DRGET
IEPRI	-16	Task not privileged	DRINIT
IERSU	-17	Vector in use	DRINIT
IETMO	-95	Time-out	DRGET
IEIEF	-97	Invalid event flag	DRINIT

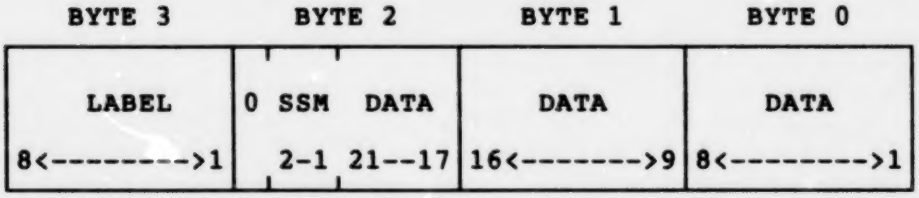


FIGURE 5-1. DRGET Word Format

APPENDIX A

INTERFACE SCHEMATIC DIAGRAM

REF	TYPE	GND	+5V
U1	74LS00	7	14
U2	74LS02	7	14
U3	74LS04	7	14
U4	74LS14	7	14
U5	74LS20	7	14
U6	74LS109	8	16
U7	74LS109	8	16
U8	74LS163	8	16
U9	MK4501N-15	14	28
U10	MK4501N-15	14	28
U11	AMI 575042-1	9	18
U12	AM26LS33	8	16

HIGHEST NUMBER USED	NOT USED
C17	
R23	
U12	U1A U1B U1C
	U2A U2B
	U4A U4B U4C

NOTES:

1. Unless otherwise specified:
   All resistors are in Ohms, ±10%, 1/8 watt.
   All capacitors are in µF, 15V.
   All voltages are in DC.

2. Device type number in the table above is for reference only. The number varies with the manufacturer.

3. Part types are abbreviated in the field of the drawing. For full part type, refer to the table above.

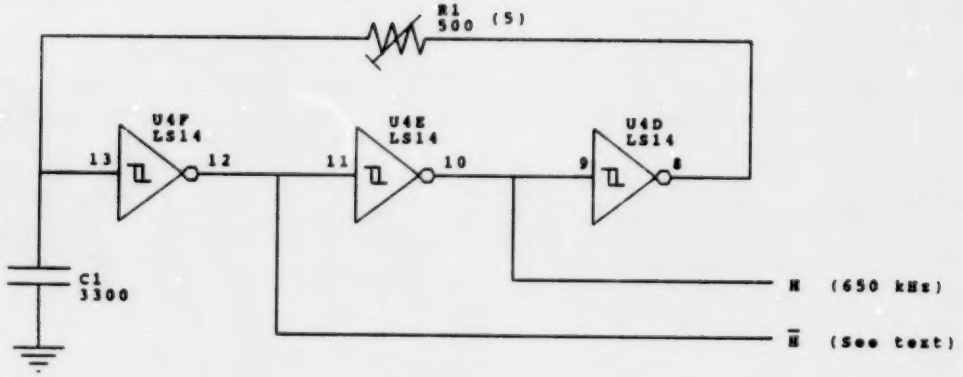
4. Capacitors C6 to C17 are all 0.1 µF and are used for decoupling between GND and Vcc for U1 to U12.

5. Adjust R1 for f = 650 kHz.

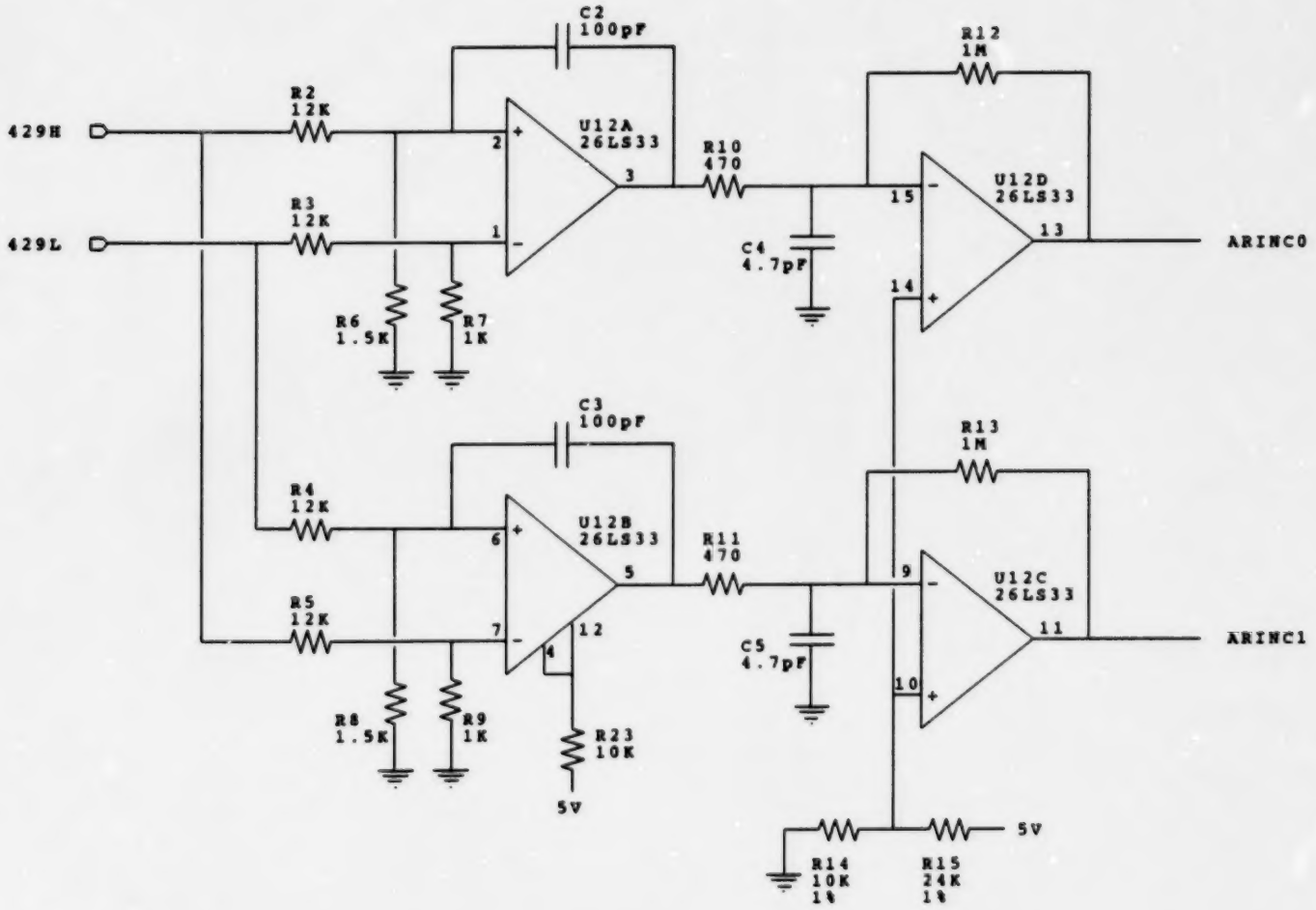
6. Pin 9 of U5 is reserved for future use.

7. Inputs of unused gates are connected to 5V through a 10K resistor (R22).

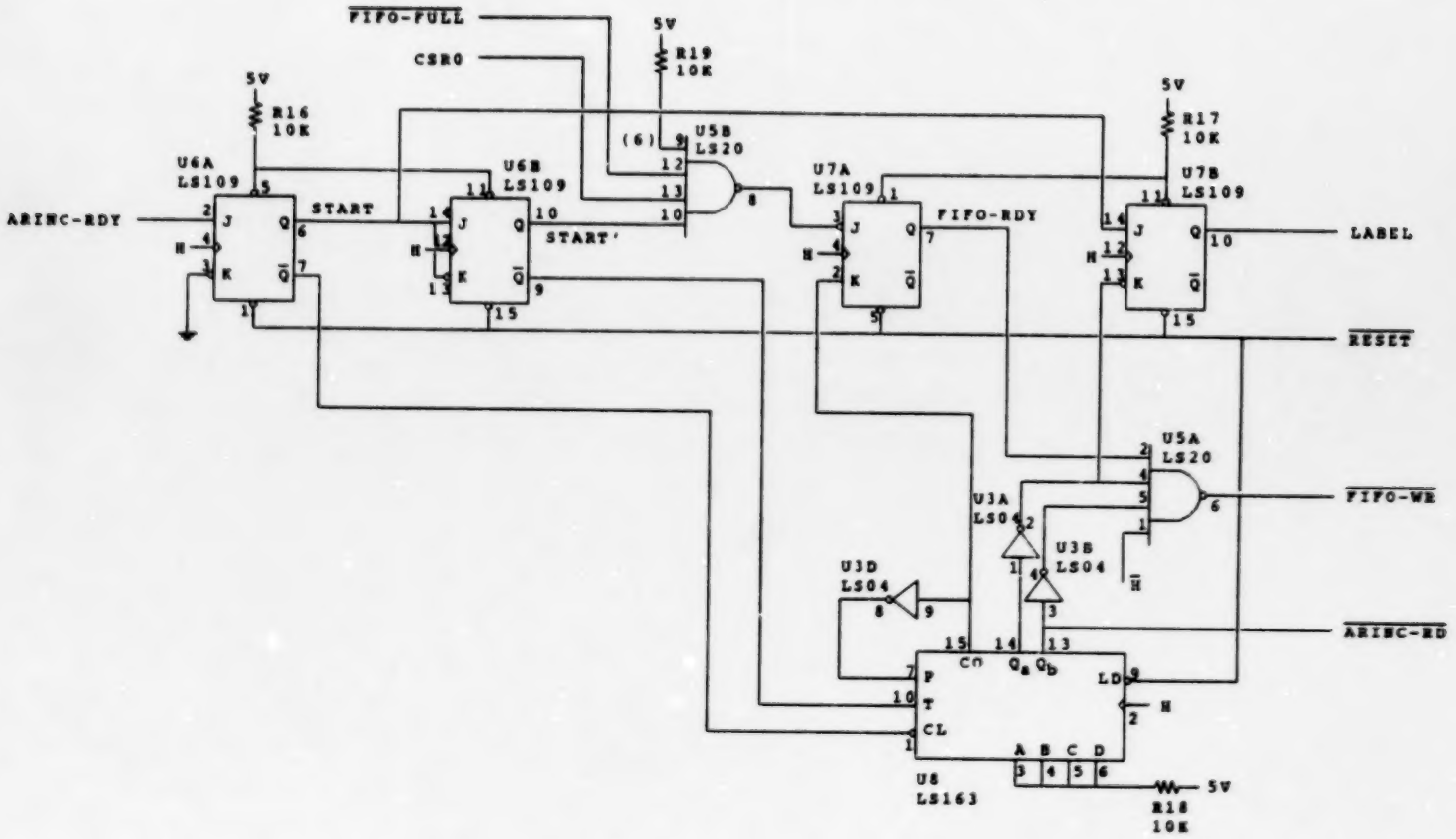
Appendix A. Schematic Diagram (Sheet 1 of 5)



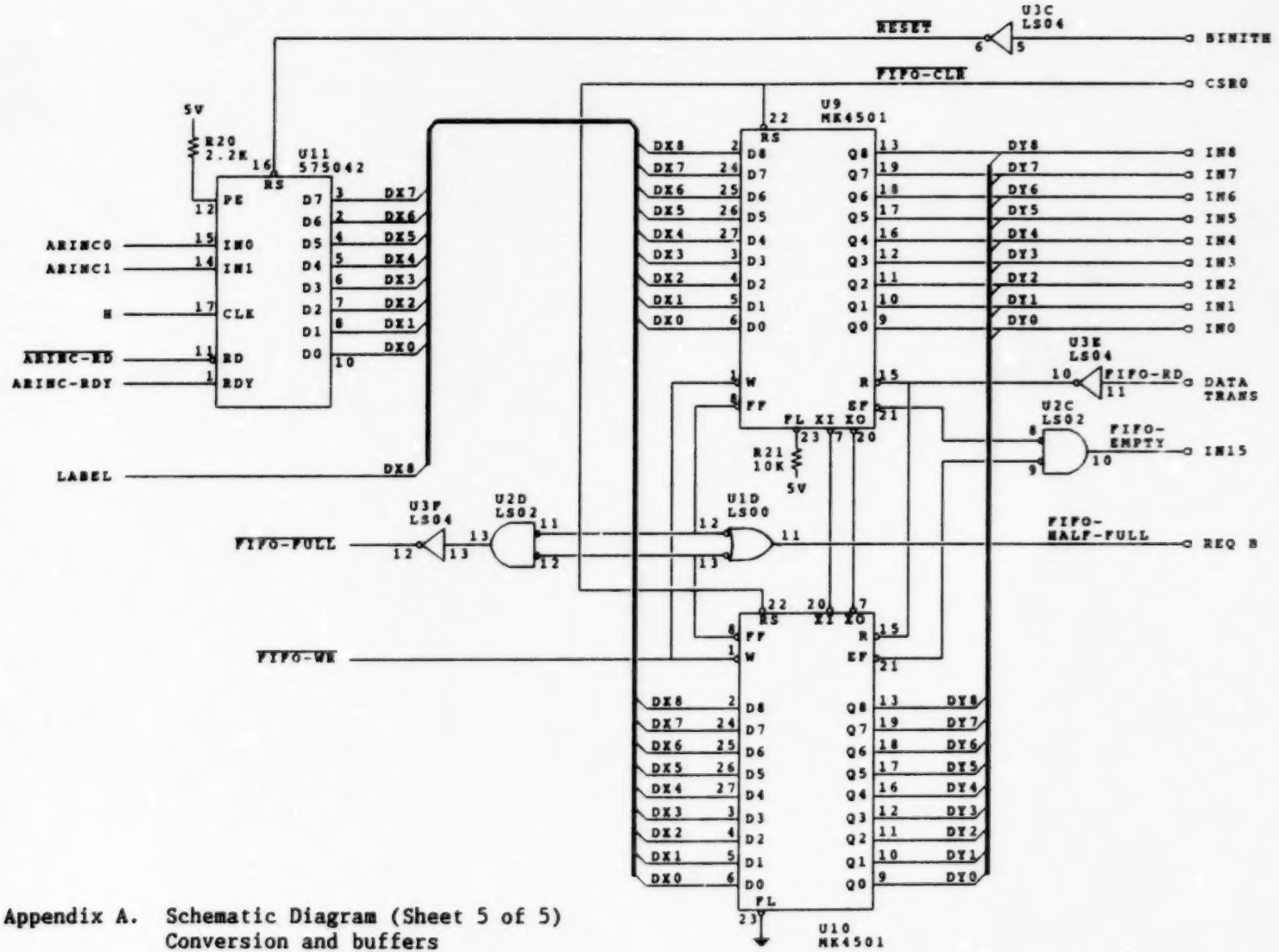
Appendix A. Schematic Diagram (Sheet 2 of 5)
Oscillator



Appendix A. Schematic Diagram (Sheet 3 of 5)
Line receivers



Appendix A. Schematic Diagram (Sheet 4 of 5)
Sequencer and control



Appendix A. Schematic Diagram (Sheet 5 of 5)
Conversion and buffers

APPENDIX B

INTERFACE DRIVER

```
        .Title  ARINC   ARINC-DR11 INTERFACE
        .Ident  /V1.2B/
        .Enabl  Lc

;+
;       File:           ARINCDR11.mac
;
;       Description:    This module is used with a DRV11 interface in
;                       conjonction with an external interface including a
;                       Litton integrated circuit to read ARINC serial data
;                       and a buffer memory (1024-byte FIFO).  The data arrives
;                       at a rate of 4kBytes/Sec in 4-bytes words.  The first
;                       byte is the label field and is used to identify the
;                       data that follows.
;
;                       All the subroutines defined herein are called from
;                       high-level languages such as Fortran and the data is
;                       placed in buffers accessed through a Fortran COMMON.
;
;       To assemble:    MAC>ARINCDR11, ARINCDR11/-SP=   -
;                       MAC>[1,1]EXEMC/ML,              -
;                       MAC>[11,10]RSXMC/PA:1,          -
;                       MAC>[1,100]ARINCDR11
;
;       To link:        TKB>Task/PR:0/-SG/-SQ = Task
;                       TKB>ARINCDR11, ARINCLUT
;                       TKB>LB:[1,54]RSX11M.stb/SS
;                       TKB>/
;                       TKB>COMMON=IOPAGE:RW
;                       TKB>EXTSCT=INTRP3:10000
;                       TKB>GBLDEF=DR$CSR:170200
;                       TKB>GBLDEF=DR$VEC:150
;                       TKB>//
;
;                       Note:   The switches "/-SG/-SQ" are VERY important.
;                               See section INTRPx.
;
;                       If the task is overlaid, the modules ARINCDR11 and
;                       ARINCLUT must be put in the root.  Those modules have
;                       not been tested with overlaid tasks.
;
;       Author:         Marc Dion
;
;       Rev     Date    Author  Description
;       ---     ----    ------  -----------
;       V1.0    Apr87   MD      Original version, never released
;       V1.1    Aug87   MD      Based on a DRV11 and a FIFO hardware buffer
;       V1.2    Nov87   MD      Add timeout support
;       V1.2B   Feb88   MD      Fine-tune timeout support
;-


;+
;       Example:
```

```
;       -------
;
;           Byte        Labels(6)               ! We ask for those 6 labels
;           Data        Labels /"330,"331,"332,"333,"334,"336/
;
;           Common /INTRP3/     Buf             ! Data written in that area
;           Byte                bBuf(4,6,1)     ! Dimensioned to our needs
;           Equivalence (Buf,bBuf)
;
;           Call DRINIT ( 2, ids )              ! Connect to the interface
;           Call DRSEL  ( Labels, 6, ids )      ! Select the labels
;           Call DRSTRT ( 1, "330, ids )        ! Start at 64 Hertz rate
;
;         5 Call DRGET  ( iPtr, iCnt, ids )     ! Now, read a buffer
;
;           Write (5,8) ((bBuf(i,j,1+iPtr/24),i=1,4),j=1,6)
;         8 Format(4o5.3)                       ! Data in bBuf(*,*,1+iPtr/24)
;
;           Goto 5                              ! Repeat
;
;         9 Call DRSTOP ( ids )                 ! When done, stop
;           Call DRDISC ( ids )                 ! and disconnect.
;-


;$DEBUG=      0              ; Include debug statements if defined

      .Mcall  ASTX$S, CINT$, CLEF$, CMKT$, DIR$, MRKT$, WTSE$
      .Mcall  HWDDF$, ITBDF$

      HWDDF$
      ITBDF$




      .Sbttl  BRKPT   --  Debugging aid macro.

;+
;     Description:    Macro to conditionally include and execute breakpoints.
;
;     Arguments:      $DEBUG  Symbol, if not defined, macro expansion is
;                             skipped.
;                     MASK    Macro-argument, if any bit set in MASK is also
;                             set in DBG.BPT, then BPT is skipped.
;                             If MASK is 0, then BPT are never skipped.
;                             Default is -1, BPT are skipped if DBG.BPT not 0
;                     DBG.BPT Variable, in psect INTRP0, to control BPT
;                             processing at run-time.
;                     DBG.CTL Variable to control execution during debugging
;                             DBG.BPT & DBG.CTL can be set via a GBLPAT.
;-
```

```
      .Macro  BRKPT   Mask=-1 ?L              ; Debugging breakpoint
      .If df  $DEBUG                          ; Debug only if $DEBUG defined
      bit     DBG.bpt,#Mask                   ; Skip breakpoint?
      bne     L                               ;  |
      Bpt                                     ; Effective in ODT & XDT
L:
      .Endc ; $DEBUG
      .Endm


      .If df  $DEBUG
      .Psect  $CODE1
DRDBG::                                       ; Special entry point for debug
      Bpt                                     ; Permanent Breakpoint
      Return
      .Endc ; $DEBUG




      .Page
      .Sbttl  DRINIT  --  Connect to vector.

;+
;     Description:    This subroutine connects to the DRV11 interface and
;                     initializes the buffers.
;
;     Fortran call:   Call DRINIT ( efl, ids )
;
;     Arguments:      EFL     I*2     Event flag used for synchronisation
;                                     Must NOT be used outside the DR/ARINC
;                                     subroutines.
;
;     Global defn:    The following symbols/arguments MUST be defined at
;                     task-build time by using the GBLDEF option.
;
;                     DR$CSR  I*2     CSR address of DRV11 interface
;                     DR$VEC  I*2     Interrupt vector of interface
;
;     Results:        IDS     I*2     (Directive) status
;                                     IE.ITS - DRxxx calls out of sequence
;                                     IE.IEF - Invalid event flag
;                                     IE.SPC - Pool area too big
;-

c..efl=       2                 ; Event flag to used for synchronization
c..ids=       4                 ; Status returned to user

      .Globl  DR$CSR, DR$VEC

      .Psect  $CODE1  Rw, I, Lcl, Rel, Con
```

```
DRINIT::
        BrkPt   000001                          ; drINIT

        bit     #st.con,Status                  ; Channel already open?
        beq     $010                            ;  |
        mov     #IE.ITS,@c..ids(r5)             ;  | Yes: return error code
        Return                                  ;  |      and abort

$010:   mov     @c..efl(r5),r4                  ; Event flag for sync
        cmp     r4,#32.                         ; Local Event flag?
        blos    $011                            ;  |
        mov     #IE.IEF,@c..ids(r5)             ;  | No: return error code
        Return                                  ;  |     and abort

$011:   cmp     #Pool.End+APR5-<$BASE&177700>,#APR5+APRsize
        blos    $012                            ; Br: Pool falls within APR5
        mov     #IE.SPC,@c..ids(r5)             ; Pool too big
        Return

$012:   mov     r4,efl                          ; Save event flag #
        mov     r4,Cl.Ef+c.leef                 ; Fill-in DPB with Efl
        mov     r4,Wt.Se+w.tsef                 ;  |
        mov     r4,Mrk.T+m.ktef                 ;  |
        mov     r4,Cmk.T+c.mkef                 ;  |

        mov     #DR$CSR,r0                      ; DRV11 base address
        tst     (r0)                            ; Crach if DRV11 not there
        mov     r0,DR.Csr                       ; Save CSR address
        add     #x..In,r0                       ; Input buffer register offset
        mov     r0,DR.In                        ; Save it.

        mov     #DR$VEC,r0                      ; DRV11 vector address
        add     #x..vB,r0                       ; Access 2nd vector: REQ B

        mov     r0,DR.con+c.inve                ; Save VEC into DBP
        mov     r0,DR.dis+c.inve                ;  |


        mov     #L$IDX,r0                       ; Index table
        mov     #256.,r1                        ;  | size of table
        mov     #-1,r2                          ;  | initial value= dont store
$040:   mov     r2,(r0)+                        ; Initialize table
        sob     r1,$040                         ;  |

        DIR$    #DR.CON                         ; Connect to DRV11

        mov     $DSW,@c..ids(r5)                ; Return status

        Return
```

```
      .Page
      .Sbttl  DRDISC  --  Disconnect from vector.

;+
;     Description:    This subroutine closes the communication channel.
;                     It disconnects from interrupt vector.
;
;     Fortran call:   "Call DRDISC ( ids )"
;
;     Arguments:      Nil
;
;     Results:        IDS     I*2     (Directive) status
;                                     IE.ITS - DRxxx calls out of sequence
;
;     Note:           If bit ST.ENA is set, a call to DRSTOP is implied,
;                     but it is good practice to explicitely call it first.
;-

d..ids=       2               ; Status returned to the user

DRDISC::
      BrkPt   000002                          ; drDISC

      bit     #st.con,Status                  ; Channel open?
      bne     $050                            ; |
      mov     #IE.ITS,@d..ids(r5)             ; | No: return error code
      Return                                  ; |     and abort

$050: DIR$    #DR.DIS

      mov     $DSW,@d..ids(r5)                ; Return status
      Return




      .Page
      .Sbttl  DRSEL   --  Select labels

;+
;     Description:    This subroutine is used to specify which of the ARINC
;                     data to store, by using the label field.
;
;     Fortran call:   Call DRSEL ( Labels, N, ids )
;
;     Arguments:      LABELS  B[n]    Byte array specifying the labels.
;                     N       I*2     Size of LABELS
;
;     Results:        IDS     I*2     (Directive) status
;                                     IE.ITS - DRxxx calls out of sequence
;-

se.lbl=  2            ; Offsets in argument list (*R5)
```

```
se.n=    4              ;
se.ids=  6              ;

DRSEL::
      Brkpt   000004                          ; drSEL

      cmpb    #ss.con,Status                  ; Call valid? (in state SS.ENA)
      beq     $060                            ;  |
      mov     #IE.ITS,@se.ids(r5)             ;  | No: return error code
      Return
$060:

      mov     #L$IDX,r0                       ; Index table
      mov     #256.,r1                        ;  | size of table
      mov     #-1,r2                          ;  | initial value= dont store
$061: mov     r2,(r0)+                        ; Initialize table
      sob     r1,$061                         ;  |


      mov     @se.n(r5),r1                    ; Argument N
      beq     $067                            ;  |
      clr     r2                              ; Offsets to store
      mov     se.lbl(r5),r0                   ; Adr of LABELS

$062: movb    (r0)+,r3                        ; Label
      bic     #^C377,r3                       ; Mask high-byte
      add     r3,r3                           ; Convert to word index

      tst     L$IDX(r3)                       ; Label already selected?
      bpl     $064                            ;  | Br: Yes

      mov     r2,L$IDX(r3)                    ; Store offset in indexed-table
      add     #4,r2                           ; Nxt offset pt nxt double-word

$064: sob     r1,$062                         ; Repeat

      mov     r2,Buf.Siz                      ; Size of buffer needed

      mov     #Pool.End+APR5-<$BASE&177700>,Buf.End
      sub     r2,Buf.End                      ; For test wrap-around

      mov     #IS.SUC,@se.ids(r5)             ; Return success
$067: Return




      .Page
      .Sbttl  DRSTRT  --  Start collecting ARINC data.

;+
;     Description:    This subroutine is used to enable data acquisition.
;                     Interrupts are enabled, entering the initial acquisi-
```

```
;                       tion state in which it is hunting for sync.  Then
;                       8 frames are acquired for initial set up of buffer.
;                       Buffer allocation is not enable.
;
;       Fortran call:   "Call DRSTRT ( Rate, Sync, ids )"
;
;       Arguments:      RATE    I*2     Rate to create buffers.  Each buffer
;                                       holds a full frame, as selected by
;                                       DRSEL. RATE is specified in 1/64th sec)
;                       SYNC    I*2     Label used as time marker.  Should be
;                                       one of the highest rate labels.
;
;       Results:        IDS     I*2     (Directive) status
;                                       IE.ITS - DRxxx calls out of sequence
;-

sa.rate=  2             ; Offsets in arguments list (*R5)
sa.sync=  4
sa.ids=   6

DRSTRT::
        Brkpt   000010                          ; drSTRT

        cmpb    #ss.con,Status                  ; Call valid? (in state SS.ENA)
        beq     $112                            ;  |
$111:   mov     #IE.ITS,@sa.ids(r5)             ;  | No: return error code
        Return
$112:

        mov     #Pool,r0                        ; Adr of 1st buffer in pool
        mov     Buf.Size,r1                     ; Size of each buffer
        beq     $111                            ;  | no label selected
        asr     r1                              ;  | in words
$115:   clr     (r0)+                           ; Initialize (Zap) buffer
        sob     r1,$115                         ;  |

        mov     #Pool+APR5-<$BASE&177700>,Buf.Cur; 1st buf in pool, Kernel mode
        mov     Buf.Cur,Buf.1st                 ; Next buffer to user
        bic     #st.1st,Status                  ;  | but not allocated to user

        mov     @sa.rate(r5),r0                 ; User specified rate
        mov     r0,Rate.Val                     ;  |
        ash     #-5,r0                          ; Setup timer value
        add     #4,r0                           ;  |
        mov     r0,MRK.T+m.ktmg                 ;  | tmg <- Rate/32 + 4
        movb    @sa.sync(r5),Sync               ; User specified sync label
        mov     #9.,Rate.Cnt                    ; Acquire at least 8 frames
        movb    #3,Iaq.Cnt                      ; IAQ for at least 2 frames

        .If df  $DEBUG
        mov     #DBG.Icnt,r0                    ; Initialize DBG.Icnt DBG.Vcnt
        mov     #8.,r1                          ;  |          DBG.Fcnt DBG.Bcnt
$120:   clr     (r0)+                           ;  |
        sob     r1,$120                         ;  |
```

```
      .Endc

      bis     #ss.spn,Status                  ; We are nov reading ARINC

      clr     Tmp.Cnt                         ; 1st, expect a label
      tstb    @DR.In                          ; Flush/ reset interrupt
      bis     #..Ena,@DR.Csr                  ; Enable FIFO & interrupts

      mov     #IS.SUC,@sa.ids(r5)             ; Return success code
      Return




      .Page
      .Sbttl  DRSTOP  --  Stop collecting ARINC data.

;+
;     Description:    This subroutine stops all activity, but does not
;                     disconnect from vector.
;
;     Fortran call:   "Call DRSTOP ( ids )"
;
;     Arguments:      None
;
;     Results:        IDS     I*2     (Directive) status
;                                     IE.ITS - DRxxx calls out of sequence
;-

so.ids=  2            ; Offsets in arguments list (*R5)

DRSTOP::
      Brkpt   000020                          ; drSTOP

      bit     #st.ena,Status                  ; Started?
      bne     $122                            ;  |
      mov     #IE.ITS,@so.ids(r5)             ;  | No: return error code
      Return
$122:

      bic     #st.spn!st.ena,Status           ; Back to state SS.CON

      bic     #..Ena,@DR.Csr                  ; Disable FIFO & interrupts

      mov     #IS.SUC,@so.ids(r5)             ; Return success code
      Return




      .Page
      .Sbttl  DRGET   --  Read a frame
```

```
;+
;     Description:    This subroutine returns a pointer to a full frame
;                     (as defined by DRSEL) of ARINC data.
;                     If the data is not in buffer already, the task will
;                     wait for it.
;
;                     The buffers/frames are accessed through the common
;                     INTRP3.
;
;     Fortran call:   "Call DRGET ( Fidx, Fsiz, ids )"
;
;     Arguments:      None
;
;     Results:        FIDX    I*2     Index in common INTRP3, in bytes,
;                                     origin 1
;                     FSIZ    I*2     Size of a frame, in bytes
;                     IDS     I*2     (Directive) status
;                                     IE.ITS - DRxxx calls out of sequence
;                                     IE.DAO - Data overrun detected, not
;                                              necessarry in this frame
;                                     IE.TMO - Timeout, no buffer returned
;                                              (FIDX & FSIZ are 0)
;-


g..idx= 2               ; User buffer
g..siz= 4               ; Size of a frame
g..ids= 6               ; Status returned to the user

DRGET::
      BrkPt   000040                            ; drGET

      bit     #st.ena,Status                    ; Started?
      bne     $202                              ;  |
      mov     #IE.ITS,@g..ids(r5)               ;  | No: return error code
      Return                                    ;  |     and abort
$202:
      bit     #st.1st,Status                    ; Buffer allocated to user?
      beq     $210                              ;  | Br: No
                                                ;  | Yes:
      add     Buf.Siz,Buf.1st                   ;  |  Release this buffer
      cmp     Buf.1st,Buf.End                   ;  |  Wrap-around?
      ble     $210                              ;  |   |
      mov     #Pool.Beg+APR5-<$BASE&177700>,Buf.1st; | Yes:
$210:

      DIR$    #CL.EF                            ; Reset sync flag

      bis     #st.req,Status                    ; Interlock with ISR

      cmp     Buf.1st,Buf.Cur                   ; Frame already in pool?
      bne     $214                              ;  |
                                                ;  | No:
```

```
        .If df  $DEBUG                          ;  |  |
        bit     #DB.tmo,DBG.ctl                 ;  |  | Skip timeout option?
        bne     $213                            ;  |  |  | Br: Yes
        .Endc                                   ;  |  |
        DIR$    #MRK.T                          ;  |  | Start timeout timer
$213:   DIR$    #WT.SE                          ;  |  | Wait from ISR
        DIR$    #CMK.T                          ;  |  | & Cancel timer
$214:
        bic     #st.req,Status                  ; Interlock with ISR

        bit     #st.tmo,Status                  ; Timeout condition?
        bne     $240                            ;  | Br: Yes

        bis     #st.1st,Status                  ; Buffer passed is alloc

        mov     Buf.1st,@g..idx(r5)             ; Pass pointer to frame buffer
        sub     #Pool.Beg+APR5-<$BASE&177700>-1,@g..idx(r5)

        mov     #IS.SUC,@g..ids(r5)             ; Assume success code
        bit     #st.ovr,Status                  ; Data overrun occured ?
        beq     $220                            ;  | Br: No
        bic     #st.ovr,Status                  ;  | Yes: reset flag
        mov     #IE.DAO,@g..ids(r5)             ;  |      and return error code
$220:
        mov     Buf.Siz,@g..siz(r5)             ; Return size of frame
        Return

$240:                                           ; Timeout:
        bic     #st.tmo,Status                  ; Reset flag

        clr     @g..idx(r5)                     ; Return no buffer
        clr     @g..siz(r5)                     ;  |
        mov     #IE.TMO,@g..ids(r5)             ; Return status
        Return

MK.AST:                                         ; AST routine for MRKT$
        bis     #st.tmo,Status
        tst     (sp)+
        ASTX$S




        .Page
        .Sbttl  DRSPND  --  Suspend operations

;+
;       Description:    Continue data acquisition, but suspend buffer/frame
;                       allocation.  Previous buffers are lost, current buffer
;                       can be accessed with DRGET.  Can be used to clear
;                       buffers.
;
;       Fortran call:   Call DRSPND ( ids )
```

```
;
;     Arguments:      None
;
;     Results:        IDS     I*2     Status
;                                     IE.ITS - DRxxx calls out of sequence
;-

sp.ids= 2             ; Offsets in argument list

DRSPND::
      Brkpt   000100                          ; drSPND

      cmpb    #ss.ena,Status                  ; Call valid?
      beq     $310                            ;  | Br: Yes, state is SS.ENA
      mov     #IE.ITS,@sp.ids(r5)             ;  | No: Return error code
      Return
$310:
      bis     #ss.spn,Status                  ; Show state

      mov     Buf.Cur,Buf.lst                 ; Forget about previous buffers
      bic     #st.lst,Status                  ;  |

      mov     #IS.SUC,@sp.ids(r5)
      Return


      .Sbttl  DRRESM  --  Resume operations

;+
;     Description:    Resume buffer/frame allocation.
;                     No IAQ involved.
;
;     Fortran call:   Call DRRESM ( ids )
;
;     Arguments:      None
;
;     Results:        IDS     I*2     Status
;                                     IE.ITS - DRxxx calls out of sequence
;-

re.ids= 2             ; Offsets in argument list

DRRESM::
      Brkpt   000200                          ; drRESM

      bit     #st.spn,Status                  ; Call valid?
      bne     $340                            ;  | Br: Yes, state is SS.SPN
      mov     #IE.ITS,@sp.ids(r5)             ;  | No: Return error code
      Return
$340:
      bic     #st.spn,Status                  ; Back to state SS.ENA

      mov     Buf.Cur,Buf.lst                 ; Forget about previous buffers
      bic     #st.lst,Status                  ;  |
```

```
        mov     #IS.SUC,@re.ids(r5)
        Return



        .Page
        .Sbttl  INTRPx  --  Kernel mapping

;+
;       Description:    INTRPx are a set of program sections (Psect) that are
;                       mapped automaticaly to Kernel APR 5 whenever the ISR
;                       or EDR is called.
;
;                       All code and data in the task that is used by those
;                       routines MUST fall within INTRPx section.
;
;                       To insure that all those Psect are consecutive in
;                       memory, the switches "/-SG/-SQ" MUST be specified to
;                       TKB (those are the defaults anyway).
;
;                       The different sections are listed below, they will be
;                       allocated in that order in memory
;
;                           Psect       Usage
;                           ------      ------
;                           INTRP0      Variables used in User & Kernel mode
;                           INTRP1      Executable code
;                           INTRP2      Table L$LUT
;                           INTRP3      Pool area, acessed also via COMMON
;                           INTRP4      Mark end of pool
;
;                       References from those routines that are not PC-relative
;                       must us the offset "APR5-<$BASE&177700>".
;                       All references outside the task image (ie to the exec)
;                       must be absolute.
;                       All (referencs by) pointers must be in Kernel space.
;-

        .Psect  INTRP0  Rw, I, Lcl, Rel, Con, Sav

$BASE::                         ; Virtual base address of Kernel APR5 mapping

APR5=   120000                  ; Base address of APR 5
APRsiz= 20000


        .Sbttl  INTRP0  --  Variables and tables


        .If df  $DEBUG
```

```
DBG.bpt::       .Word   0       ; Skip breakpoints if non-zero
DBG.ctl::       .Word   0       ; Debug control:
 DB.lbl0= 1                     ;   Use label 0 data field to count frames
 DB.tmo=  2                     ;   Bypass timeout processing
DBG.Icnt:       .Blkw   2       ; Count interrupts
DBG.Wcnt:       .Blkw   2       ; Count 32-bit data messages
DBG.Fcnt:       .Blkw   2       ; Count frames
DBG.Bcnt:       .Blkw   2       ; Count buffer as they are created
      .Endc

Efl:  .Blkw   1                 ; Event flag # for synchronization

Status:         .Word   0       ; General status
    st.ovr  =  40000            ;   Bit 14: RxBuf overflow, character lost
    st.fhe  =  20000            ;       13: Hardware error (parity, etc)
    st.tmo  =  10000            ;       12: Timeout on read
    st.1st  =   1000            ;        9: 1st buffer allocated to user
    st.req  =    400            ;        8: User requests buffer/frame
    st.spn  =      4            ;        2: Int Ena, suspend frame alloc
    st.ena  =      2            ;        1: Int Ena, gathering ARINC data
    st.con  =      1            ;        0: Channel open/ connected to Int.
                                ;   Lov byte defines states:
    ss.spn  = 7                 ;       Suspend: keep sync, no frame alloc
    ss.ena  = 3                 ;       Enable: keep sync, auto frame alloc
    ss.con  = 1                 ;       Connected to the DRV11, no activity
    ss.dis  = 0                 ;       Not connected to the DRV11

DR.Csr:         .Blkw   1       ; DRV11 Control/Status register address
    x..Csr  = 0                 ;                                 offset to CSR
    ..ReqB  = 100000            ;   Req B - Interrupt request on J2 (input)
   ;..ReqA  =    200            ;   Req A - Interrupt request on J1 (output)
   ;..IeA   =    100            ;   INT ENB A - Interrupt enable bit
    ..IeB   =     40            ;   INT ENB B - Interrupt enable bit
   ;..CSR1  =      2            ;   CSR1 - User defined pin on J1 (output)
    ..CSR0  =      1            ;   CSR0 - User defined pin on J2 (input)
                                ;
    ..IniL  = ..CSR0            ;   Clear FIFO buffer memory (active lov)
    ..Ena   = ..IeB ! ..IniL    ;   Enable FIFO & interrupts


DR.In:          .Blkw   1       ; Digital input buffer address
    x..In   = 4                 ;                       offset to CSR
    ..Empty = 100000            ;   Bit 15: Last byte read, FIFO nov empty
    ..Label =    400            ;        8: Lov byte is a label
    ;.Data  =    377            ;      7-0: data


x..vA =         0               ; Vector offset from base vector: Req A (out)
x..vB =         4               ;                                 Req B (in)


Rate.Val:       .Blkw   1       ; Sampling rate specified by user (in 1/64 sec)
Rate.Cnt:       .Blkw   1       ; Counts frames from 0 to RATE.VAL
```

```
Buf.Cur:        .Blkw   1       ; Buffer being filled
Buf.Siz:        .Blkw   1       ; Size of data buffers
Buf.1st:        .Blkw   1       ; 1st buffer to return to user
Buf.End:        .Blkw   1       ; End of pool area = <Pool.End-Buf.Siz>

Tmp.area:                       ; Tmp buffer to read ARINC from FIFO
Tmp.Bf:         .Blkb   4       ;   Buffer for current 32-bit word being packed
   .Label = 3                   ;       Byte   3 - Label
   .Data  = 0                   ;       Byte 2:0 - 3 byte of data
     .Low = 0                   ;                  Least significant byte
     .Mid = 1                   ;                       to
     .Hi  = 2                   ;                  Most significant byte
Tmp.Cnt:        .Blkw   1       ;   Count bytes to sync with label
Tmp.DRin:       .Blkw   1       ;   Word read from DRV11:  low=data  hi=ctl

Iaq.cnt:        .Blkb   1       ; Initial acquisition count (hunt for SYNC)
Sync:           .Blkb   1       ; Sync label
      .Even

L$IDX:          .Blkw   256.    ; Index by label: Offset to store label, or -1



      .Page
      .Sbttl  DR$ISR  --  DR Interrupt Service Routine.

;+
;     Description:    This is the Interrupt Service Routine (ISR) entered
;                     every time the FIFO buffer is half-full.
;
;     Registers:      On entry, R5 points the fork block in the ITB and R4 is
;                     free to use.  R0-R3 must to saved and restored.
;
;                     At fork-level, R4-R5 are preserved across the call and
;                     all registers are free to use.
;                     The first instruction of the fork routine must be
;                     "CLR @R3".
;
;     Return:         Return (see description of CINT$ directive)
;-


      .Psect  INTRP1  Rw, I, Lcl, Rel, Con, Sav
DR$ISR::
      bic     #..IeB,@DR.Csr                  ; Disable interrupts

      Call    @#$FORK2        ;  FORK level   ; All registers free
      clr     @r3                             ; Declare fork block free


      .If df  $DEBUG
```

```
        add     #1,DBG.Icnt                     ; Count interrupts
        adc     DBG.Icnt+2                      ;  |
        .Endc

        BrkPt   000400                          ; dr$ISR

$400:
        mov     @DR.In,Tmp.DRin                 ; Read DRV11 data

        mov     Tmp.Cnt,r0                      ; Label expected?
        bmi     $420                            ;  | Br: No, data expected

                                                ; Label expected:
        bit     #..Label,Tmp.DRin               ; Is this a label?
        beq     $412                            ;  | Br: No, error

        clr     r3                              ; Force hi-byte to 0
        bisb    Tmp.DRin,r3                     ; Copy byte, hi-byte unchanged
        movb    L$LUT+APR5-<$BASE&177700>(r3),Tmp.Bf+.Label
                                                ; Reverse bit order

        mov     #-3,Tmp.Cnt                     ; Set to count 3 data bytes


DR$NEXT:                                        ; Read DRV11 again if not empty

        Assume  ..Empty,100000
        tst     Tmp.DRin                        ; FIFO empty?
        bpl     $400                            ;  | Br: No, process next word

        mtps    #PR0                            ; Lock interrupts
        bis     #..IeB,@DR.Csr                  ; Enable interrupts

        Assume  ..ReqB,100000
        tst     @DR.Csr                         ; Interrupt already received?
        bpl     $407                            ;  | Br: No

        bic     #..IeB,@DR.Csr                  ; Disable interrupts
        mtps    #PR0                            ; Unlock interrupts
        br      $400                            ; Do it again Sam!

$407: Return                                    ; End-of-interrupt


$410: clr       Tmp.Cnt                         ; Wait for a label
$412: bis       #st.fhe,Status                  ; Indicate hardware error
      br        DR$NEXT


$420:                                           ; Data expected:
        bit     #..Label,Tmp.DRin               ; Is this data (not a label)?
        bne     $410                            ;  | Br: no, error

        movb    Tmp.DRin,Tmp.Bf+3+APR5-<$BASE&177700>(r0)
```

```
                                                ; Pack in TMP.BF

        inc     Tmp.Cnt                         ; Packing (4-bytes) completed?
        bne     DR$NEXT                         ;  | Br: No, more data to come


New.word:                                       ; We have received a 32-bit
word

        .If df  $DEBUG
        add     #1,DBG.Wcnt                     ; Count 32-bit words
        adc     DBG.Wcnt+2                      ;  |
        .Endc

        Brkpt   001000                          ; New word


        tstb    Tmp.Bf+.Data+.Hi                ; Test sign bit
        bmi     $412                            ;  | Br: bit set= h/w error

        tstb    Iaq.Cnt                         ; Initial Acquisition?
        beq     $502                            ;  | Br: No

        ; Initial acquisition phase (hunting for SYNC)

        cmpb    Tmp.Bf+.Label,Sync              ; SYNC label?
        bne     DR$NEXT                         ;  | Br: No, wait for it

        decb    Iaq.Cnt                         ; IAQ phase completed?
        bne     DR$NEXT                         ;  | Br: No

        ; Compute index

$502:   clr     r4                              ; Force hi-byte to 0
        bisb    Tmp.Bf+.Label,r4                ; Copy Label, hi-byte unchanged
        add     r4,r4                           ; Compute index in L$IDX
        mov     L$IDX+APR5-<$BASE&177700>(r4),r4; Get offset for BUF.CUR
        bmi     $510                            ; Br: Dont store, chk for sync

        cmpb    Tmp.Bf+.Label,Sync              ; SYNC label?
        beq     $530                            ;  | Br: Yes


        ; Store data, label is not SYNC
$504:   ; Store data, label is SYNC, but RATE.CNT not exhausted or state is SPN

        add     Buf.Cur,r4                      ; Index current buffer

        mov     Tmp.Bf,(r4)+                    ; Store ARINC: mid & lov data
        mov     Tmp.Bf+2,(r4)                   ;              label & hi data

        br      DR$NEXT
```

```
      ; Do not store data, but still check for SYNC
$510:
      cmpb    Tmp.Bf+.Label,Sync              ; SYNC label?
      bne     DR$NEXT                         ;  | Br: No, EOI

      dec     Rate.Cnt                        ; Time for a new buffer?
      bne     DR$NEXT                         ;  | No

      br      New.Frame


      ; Store data, label is SYNC
$530:
      dec     Rate.Cnt                        ; Time for a new buffer?
      bne     $504                            ;  | No: store frame same buf


      ; Starting a new frame, close current buffer and setup a new one.
      ; Will not reach this point in state SS.SPN

New.Frame:
      .If df  $DEBUG
      add     #1,DBG.Fcnt                     ; Count frames
      adc     DBG.Fcnt+2                      ;  |
      .Endc

      mov     Rate.Val,Rate.Cnt               ; Reset RATE.CNT

      Brkpt   002000                          ; New.Frame

      mov     Buf.Siz,r0
      mov     Buf.Cur,r1

      .If df  $DEBUG
      bit     #DB.lbl0,DBG.ctl                ; Special use of label 0?
      beq     $601                            ;  | Br: No
      mov     L$IDX,r2                        ; Index for label 0
      bmi     $601                            ;  | Label not selected
      add     r1,r2                           ; Index current buffer
      mov     DBG.Bcnt,(r2)+                  ; Copy only the 3 MSBytes
      movb    DBG.Bcnt+2,(r2)                 ;  |
$601:
      .Endc

      bit     #st.spn,Status                  ; Create new buffer only if
      beq     $602                            ; state is SS.ENA or if state
      bit     #st.req,Status                  ; is SS.SPN and a request is
      beq     $620                            ; pending (ST.REQ).


                                              ; Create & init new buffer
$602:
      mov     r1,r3                           ; Pt next buffer
      add     r0,r3                           ;  |
```

```
        cmp     r3,Buf.End                      ; Wrap-around?
        ble     $604                            ;  | Br: No
        mov     #Pool.Beg+APR5-<$BASE&177700>,r3;  | Yes, wrap-around
$604:
        cmp     r3,Buf.1st                      ; Overflow?
        bne     $610                            ;  | Br: No
        bis     #st.ovr,Status                  ;  | Yes, set flag
        br      $620                            ;  |      and use same buffer

$610:                                           ; OK to create nev buffer
        .If df  $DEBUG
        add     #1,DBG.Bcnt                     ; Count buffers
        adc     DBG.Bcnt+2                      ;  |
        .Endc

        mov     r3,r2                           ; Preserve R3
        asr     r0                              ; Convert to word count
$614:   mov     (r1)+,(r2)+                     ; Init new buffer with old one
        sob     r0,$614                         ;  |

        mov     r3,Buf.Cur                      ; Nev buffer is nov current


$620:   tst     r4                              ; Index to store SYNC
        bmi     $640                            ;  | Br: do not store

        add     Buf.Cur,r4                      ; Index current buffer
        mov     Tmp.Bf,(r4)+                    ; Store mid & lov data byte
        mov     Tmp.Bf+2,(r4)                   ;       label & high data byte

$640:
        bit     #st.req,Status                  ; User requests frame?
        beq     $677                            ;  | Br: No

        bic     #st.req,Status

                                                ; Wake up user
        mov     Efl,r0                          ; Sync event flag
        mov     x.tcb-x.fork(r5),r5             ; User TCB
        Call    @#$SETF                         ; Set event flag

$677:   jmp     DR$NEXT




        .Page
        .Sbttl  DR$EDR  --  DR Enable Disable Routine

;+
;       Description:    DR$EDR is called whenever CINT$ is executed for
;                       connecting or disconnecting to the vector.
```

```
;                       Control is dispatched to DR$CON or DR$DIS.
;
;                       DR$CON  merely sets the STATUS word.  Activity wont
;                               start until DRSTART is called.
;                       DR$DIS  resets the status word and disable interrupts.
;
;       Call:           From the executive, in Kernel mode, during the
;                       execution of the CINT$ directive.
;
;       Return:         "Rts pc"
;
;       Registers:      R0, R2, R3 are free.  Others must be preserved.
;
;       Arguments:      R1 - Point the ITB
;                       CC - The routine is called during a "connect to inter-
;                            rupt vector".
;                       CS - Called during a "disconnect from interrupt vector"
;-

        .Psect  INTRP1

DR$EDR::
        BrkPt   004000                          ; dr$EDR

        mov     #0,@DR.Csr                      ; Disable DRV11 carry unchanged

        bcs     DR$DIS

DR$CON:
        mov     #ss.cca,Status                  ; Channel nov open

        Return

DR$DIS:
        clr     Status                          ; Channel nov close

        Return




        .Page
        .Sbttl  INTRP3  --  Buffer pool area

        .Psect  INTRP3  Rw, D, Gbl, Rel, Ovr, Sav

Pool:
Pool.Beg:
        .If ndf $DEBUG
        .Blkb   1024.                   ; Pool area: minimum size to guaranty
                                        ;            at least 2 buffers with
                                        ;            all labels selected.
                                        ; Use the EXTSCT option with TKB to
```

```
                                        ; increase its size.
        .Iff
        .Blkb   100                     ; User smaller pool during debugging
        .Endc

        .Psect  INTRP4
Pool.End=       .                       ; End of pool



        .Page
        .Sbttl  $IDATA  --  Variables and impure literals
        .Psect  $IDATA  Rw, D, Lcl, Rel, Con


DR.con: CINT$ , $Base, DR$Isr, DR$Edr, PR5      ; Connect DR
DR.dis: CINT$                                   ; Disconnect DR


                                                ; Note: EF# fill-in at run time
Cl.Ef:  CLEF$                                   ; Clear event flag
Wt.Se:  WTSE$                                   ; Wait for event flag
Mrk.T:  MRKT$ , 1, 2, MK.AST                    ; Timer x sec, AST
Cmk.T:  CMKT$ , MK.AST                          ; Cancel timer

        .End
```

```
        .Title  LUT     - ARINC LABEL'S LUT
        .Ident  /V1.0/
        .Enabl  Lc

;+
;       File:           ARINCLUT.mac
;
;       Description:    This code produce a look-up table for label fields.
;                       It is used to reverse quickly a bit patterns (8 bits).
;                       The table is 256. bytes long.
;
;                       It is built at the beginning of the Psect INTRP2.
;                       See ARINCDR11 for description of PSECT usage.
;
;       To compile:     MAC ARINCLUT, ARINCLUT/-SP= ARINCLUT
;
;       To link:        See ARINCDR11.mac
;
;       Author:         Marc Dion
;
;       Rev     Date    Author  Description
;       ---     ----    ------  -----------
;       V1.0            MD      Original version
;-

        .Psect  INTRP2  Rw, D, Gbl, Rel, Ovr, Sav

        .Globl  L$LUT
L$LUT::

x =     0

        .Nlist
        .Rept   256.

y=      0
y=      y ! <   x & ^B 10000000 / ^B 10000000 >
y=      y ! <   x & ^B 01000000 / ^B   100000 >
y=      y ! <   x & ^B 00100000 / ^B     1000 >
y=      y ! <   x & ^B 00010000 / ^B       10 >
y=      y ! <   x & ^B 00001000 * ^B       10 >
y=      y ! <   x & ^B 00000100 * ^B     1000 >
y=      y ! <   x & ^B 00000010 * ^B   100000 >
y=      y ! <   x & ^B 00000001 * ^B 10000000 >

.Byte y

x=      x + 1
        .Endr

        .End
```

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

1. ORIGINATOR (the name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Establishment sponsoring a contractor's report, or tasking agency, are entered in section 8.)

Defence Research Establishment Ottawa
National Defence Headquarters
Ottawa, Ontario
Canada K1A 0Z4

2. SECURITY CLASSIFICATION (overall security classification of the document including special warning terms if applicable)

UNCLASSIFIED

3. TITLE (the complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S,C,R or U) in parentheses after the title.)

Description of an ARINC Bus Interface for an LSI-11 minicomputer(U)

4. AUTHORS (Last name, first name, middle initial)

Dion, Marc

5. DATE OF PUBLICATION (month and year of publication of document)

December 88

6a. NO. OF PAGES (total containing information. Include Annexes, Appendices, etc.)

47

6b. NO. OF REFS (total cited in document)

4

7. DESCRIPTIVE NOTES (the category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

Technical Note

8. SPONSORING ACTIVITY (the name of the department project office or laboratory sponsoring the research and development. Include the address.)

9a. PROJECT OR GRANT NO. (if appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant)

41LJ

9b. CONTRACT NO. (if appropriate, the applicable number under which the document was written)

10a. ORIGINATOR'S DOCUMENT NUMBER (the official document number by which the document is identified by the originating activity. This number must be unique to this document.)

DREO TECHNICAL NOTE 88-30

10b. OTHER DOCUMENT NOS. (Any other numbers which may be assigned this document either by the originator or by the sponsor)

11. DOCUMENT AVAILABILITY (any limitations on further dissemination of the document, other than those imposed by security classification)

(X) Unlimited distribution
( ) Distribution limited to defence departments and defence contractors; further distribution only as approved
( ) Distribution limited to defence departments and Canadian defence contractors; further distribution only as approved
( ) Distribution limited to government departments and agencies; further distribution only as approved
( ) Distribution limited to defence departments; further distribution only as approved
( ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in 11) is possible, a wider announcement audience may be selected.)

UNCLASSIFIED

SECURITY CLASSIFICATION OF FORM

13. ABSTRACT (a brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), (R), or (U). It is not necessary to include here abstracts in both offical languages unless the text is bilingual).

An interface between the ARINC bus and an LSI-11 mini-computer was designed and built at DREO. The interface accepts serial data from a single twisted pair of cables which meet the ARINC 429 interface protocol specifications. The data is transfered in the minicomputer memory through a DEC standard DRV11 parallel interface. An interface driver is provided to control the operation of the interface.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus. e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus-identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

ARINC
ARINC BUS
Q-BUS
Q-BUS INTERFACE
LSI-11 INTERFACE
ARINC BUS INTERFACE